How does IPv4 Subnetting Work?

Asked 10 years, 2 months ago Active 2 months ago Viewed 155k times

439

471

This is a Canonical Question about IPv4 Subnets.

Related:

- How does IPv6 subnetting work and how does it differ from IPv4 subnetting?

How does Subnetting Work, and How do you do it by hand **or in your head?** Can someone explain both conceptually and with several examples? Server Fault gets lots of subnetting homework questions, so we could use an answer to point them to on Server Fault itself.

- If I have a network, how do I figure out how to split it up?
- If I am given a netmask, how do I know what the network Range is for it?
- Sometimes there is a slash followed by a number, what is that number?
- Sometimes there is a subnet mask, but also a wildcard mask, they seem like the same thing but they are different?
- Someone mentioned something about knowing binary for this?

networking subnet tcpip ipv4

edited Dec 22 '18 at 18:30

community wiki
20 revs, 11 users 40%
Kyle Brandt

8 Answers

648

IP subnets exist to allow routers to choose appropriate destinations for packets. You can use IP subnets to break up larger networks for logical reasons (firewalling, etc), or physical need (smaller broadcast domains, etc).

Simply put, though, IP routers use your IP subnets to make routing decisions. Understand how those decisions work, and you can understand how to plan IP subnets.

Counting to 1

If you are already fluent in binary (base 2) notation you can skip this section.

For those of you who are left: Shame on you for not being fluent in binary notation!

Yeah-- that may be a bit harsh. It's really, really easy to learn to count in binary, and to learn shortcuts to convert binary to decimal and back. You really should know how to do it.

Counting in binary is so simple because you only have to know how to count to 1!

Think of a car's "odometer", except that unlike a traditional odometer each digit can only count up to 1 from 0. When the car is fresh from the factory the odometer reads "00000000".
When you've driven your first mile the odometer reads "00000001". So far, so good.

When you've driven your second mile the first digit of the odometer rolls back over to "0" (since it's maximum value is "1") and the second digit of the odometer rolls over to "1", making the odometer read "00000010". This looks like the number 10 in decimal notation, but it's actually 2 (the number of miles you've driven the car so far) in binary notation.

When you've driven the third mile the odometer reads "00000011", since the first digit of the odometer turns again. The number "11", in binary notation, is the same as the decimal number 3.

Finally, when you've driven your fourth mile both digits (which were reading "1" at the end of the third mile) roll back over to zero position, and the 3rd digit rolls up to the "1" position, giving us "00000100". That's the binary representation of the decimal number 4.

You can memorize all of that if you want, but you really only need to understand how the little odometer "rolls over" as the number it's counting gets bigger. It's exactly the same as a traditional decimal odometer's operation, except that each digit can only be "0" or "1" on our fictional "binary odometer".

To convert a decimal number to binary you could roll the odometer forward, tick by tick, counting aloud until you've rolled it a number of times equal to the decimal number you want to convert to binary. Whatever is displayed on the odometer after all that couting and rolling would be the binary representation of the decimal number you counted up to.

Since you understand how the odometer rolls forward you'll also understand how it rolls backward, too. To convert a binary number displayed on the odometer back to decimal you could roll the odometer back one tick at a time, counting aloud until the odometer reads "00000000". When all that counting and rolling is done, the last number you say aloud would be the decimal representation of the binary number the odometer started with.

Converting values between binary and decimal this way would be very tedious. You could do it, but it wouldn't be very efficient. It's easier to learn a little algorithm to do it faster.

A quick aside: Each digit in a binary number is known as a "bit". That's "b" from "binary" and "it" from "digit". A bit is a binary digit.

Converting a binary number like, say, "1101011" to decimal is a simple process with a handy little algorithm.

Start by counting the number of bits in the binary number. In this case, there are 7. Make 7 divisions on a sheet of paper (in your mind, in a text file, etc) and begin filling them in from right to left. In the rightmost slot, enter the number "1", because we'll always start with "1". In the next slot to the left enter double the value in the slot to the right (so, "2" in the next one, "4" in the next one) and continue until all the slots are full. (You'll end up memorizing these numbers, which are the powers of 2, as you do this more and more. I'm alright up to 131,072 in my head but I usually need a calculator or paper after that).

So, you should have the following on your paper in your little slots.

```
  64    |    32    |    16    |    8    |    4    |    2    |    1    |
```

Transcribe the bits from the binary number below the slots, like so:

```
 64    |    32    |    16    |    8    |    4    |    2    |    1    |
  1          1          0         1         0         1         1
```

Now, add some symbols and compute the answer to the problem:

```
 64    |    32    |    16    |    8    |    4    |    2    |    1    |
x 1        x 1        x 0       x 1       x 0       x 1       x 1
---        ---        ---       ---       ---       ---       ---
       +          +          +         +         +         +         =
```

Doing all the math, you should come up with:

```
 64    |    32    |    16    |    8    |    4    |    2    |    1    |
x 1        x 1        x 0       x 1       x 0       x 1       x 1
---        ---        ---       ---       ---       ---       ---
 64    +    32    +     0    +    8    +    0    +    2    +    1    =   107
```

That's got it. "1101011" in decimal is 107. It's just simple steps and easy math.

Converting decimal to binary is just as easy and is the same basic algorithm, run in reverse.

Say that we want to convert the number 218 to binary. Starting on the right of a sheet of paper, write the number "1". To the left, double that value (so, "2") and continue moving toward the left of the paper doubling the last value. If the number you are about to write is greater than the number being converted stop writing. otherwise, continue doubling the prior number and writing. (Converting a big number, like 34,157,216,092, to binary using this algorithm can be a bit tedious but it's certainly possible.)

So, you should have on your paper:

```
 128    |    64    |    32    |    16    |    8    |    4    |    2    |    1
|
```

You stopped writing numbers at 128 because doubling 128, which would give you 256, would be large than the number being converted (218).

Beginning from the leftmost number, write "218" above it (128) and ask yourself: "Is 218 larger than or equal to 128?" If the answer is yes, scratch a "1" below "128". Above "64", write the result of 218 minus 128 (90).

Looking at "64", ask yourself: "Is 90 larger than or equal to 64?" It is, so you'd write a "1" below "64", then subtract 64 from 90 and write that above "32" (26).

When you get to "32", though, you find that 32 is not greater than or equal to 26. In this case, write a "0" below "32", copy the number (26) from above 32" to above "16" and then continue asking yourself the same question with the rest of the numbers.

When you're all done, you should have:

```
 218        90         26         26        10        2         2         0
 128    |   64    |    32    |    16    |    8    |    4    |    2    |    1
|
   1         1          0          1         1         0         1         0
```

The numbers at the top are just notes used in computation and don't mean much to us. At the bottom, though, you see a binary number "11011010". Sure enough, 218, converted to binary, is

"11011010".

Following these very simple procedures you can convert binary to decimal and back again w/o a calculator. The math is all very simple and the rules can be memorized with just a bit of practice.

Splitting Up Addresses

Think of IP routing like pizza delivery.

When you're asked to deliver a pizza to "123 Main Street" it's very clear to you, as a human, that you want to go to the building numbered "123" on the street named "Main Street". It's easy to know that you need to go to the 100-block of Main Street because the building number is between 100 and 199 and most city blocks are numbered in hundreds. You "just know" how to split the address up.

Routers deliver packets, not pizza. Their job is the same as a pizza driver: To get the cargo (packets) as close to the destination as possible. A router is connected to two or more IP subnets (to be at all useful). A router must examine destination IP addresses of packets and break those destination addresses up into their "street name" and "building number" components, just like the pizza driver, to make decisions about delivery.

Each computer (or "host") on an IP network is configured with a unique IP address and subnet mask. That IP address can be divided up into a "building number" component (like "123" in the example above) called the "host ID" and a "street name" component (like "Main Street" in the example above) called the "network ID". For our human eyes, it's easy to see where the building number and the street name are in "123 Main Street", but harder to see that division in "10.13.216.41 with a subnet mask of 255.255.192.0".

IP routers "just know" how to split up IP addresses into these component parts to make routing decisions. Since understanding how IP packets are routed hinges on understanding this process we need to know how to break up IP addresses, too. Fortunately, extracting the host ID and the network ID out of an IP address and subnet mask is actually pretty easy.

Start by writing out the IP address in binary (use a calculator if you haven't learned to do this in your head just yet, but make a note learn how to do it-- it's really, really easy and impresses the opposite sex at parties):

```
       10.      13.     216.      41
 00001010.00001101.11011000.00101001
```

Write out the subnet mask in binary, too:

```
      255.     255.     192.       0
 11111111.11111111.11000000.00000000
```

Written side-by-side, you can see that the point in the subnet mask where the "1's" stop "lines up" to a point in the IP address. That's the point that the network ID and the host ID split. So, in this case:

```
       10.      13.     216.      41
 00001010.00001101.11011000.00101001 - IP address
 11111111.11111111.11000000.00000000 - subnet mask
 00001010.00001101.11000000.00000000 - Portion of IP address covered by 1's in
 subnet mask, remaining bits set to 0
 00000000.00000000.00011000.00101001 - Portion of IP address covered by 0's in
 subnet mask, remaining bits set to 0
```

Routers use the subnet mask to "mask out" the bits covered by 1's in the IP address (replacing the

bits that are not "masked out" with 0's) to extract the network ID:

```
      10.      13.     192.       0
00001010.00001101.11000000.00000000 – Network ID
```

Likewise, by using the subnet mask to "mask out" the bits covered by 0's in the IP address (replacing the bits that are not "masked out" with 0's again) a router can extract the host ID:

```
       0.       0.      24.      41
00000000.00000000.00011000.00101001 – Portion of IP address covered by 0's in
subnet mask, remaining bits set to 0
```

It's not as easy for our human eyes to see the "break" between the network ID and the host ID as it is between the "building number" and the "street name" in physical addresses during pizza delivery, but the ultimate effect is the same.

Now that you can split up IP addresses and subnet masks into host ID's and network ID's you can route IP just like a router does.

More Terminology

You're going to see subnet masks written all over the Internet and throughout the rest of this answer as (IP/number). This notation is known as "Classless Inter-Domain Routing" (CIDR) notation. "255.255.255.0" is made up of 24 bits of 1's at the beginning, and it's faster to write that as "/24" than as "255.255.255.0". To convert a CIDR number (like "/16") to a dotted-decimal subnet mask just write out that number of 1's, split it into groups of 8 bits, and convert it to decimal. (A "/16" is "255.255.0.0", for instance.)

Back in the "old days", subnet masks weren't specified, but rather were derived by looking at certain bits of the IP address. An IP address starting with 0 - 127, for example, had an implied subnet mask of 255.0.0.0 (called a "class A" IP address).

These implied subnet masks aren't used today and I don't recommend learning about them anymore unless you have the misfortune of dealing with very old equipment or old protocols (like RIPv1) that don't support classless IP addressing. I'm not going to mention these "classes" of addresses further because it's inapplicable today and can be confusing.

Some devices use a notation called "wildcard masks". A "wildcard mask" is nothing more than a subnet mask with all 0's where there would be 1's, and 1's where there would be 0's. The "wildcard mask" of a /26 is:

```
11111111.11111111.11111111.11000000 – /26 subnet mask
00000000.00000000.00000000.00111111 – /26 "wildcard mask"
```

Typically you see "wildcard masks" used to match host IDs in access-control lists or firewall rules. We won't discuss them any further here.

How a Router Works

As I've said before, IP routers have a similar job to a pizza delivery driver in that they need to get their cargo (packets) to its destination. When presented with a packet bound for address 192.168.10.2, an IP router needs to determine which of its network interfaces will best get that packet closer to its destination.

.

Let's say that you are an IP router, and you have interfaces connected to you numbered:

- Ethernet0 - 192.168.20.1, subnet mask /24
- Ethernet1 - 192.168.10.1, subnet mask /24

If you receive a packet to deliver with a destination address of "192.168.10.2", it's pretty easy to tell (with your human eyes) that the packet should be sent out the interface Ethernet1, because the Ethernet1 interface address corresponds to the packet's destination address. All the computers attached to the Ethernet1 interface will have IP addresses starting with "192.168.10.", because the network ID of the IP address assigned to your interface Ethernet1 is "192.168.10.0".

For a router, this route selection process is done by building a routing table and consulting the table each time a packet is to be delivered. A routing table contains network ID and destination interface names. You already know how to obtain a network ID from an IP address and subnet mask, so you're on your way to building a routing table. Here's our routing table for this router:

- Network ID: 192.168.20.0 (11000000.10101000.00010100.00000000) - 24 bit subnet mask - Interface Ethernet0
- Network ID: 192.168.10.0 (11000000.10101000.00001010.00000000) - 24 bit subnet mask - Interface Ethernet1

For our incoming packet bound for "192.168.10.2", we need only convert that packet's address to binary (as humans-- the router gets it as binary off the wire to begin with) and attempt to match it to each address in our routing table (up to the number of bits in the subnet mask) until we match an entry.

- Incoming packet destination: 11000000.10101000.00001010.00000010

Comparing that to the entries in our routing table:

```
11000000.10101000.00001010.00000010 – Destination address for packet
11000000.10101000.00010100.00000000 – Interface Ethernet0
!!!!!!!!.!!!!!!!!.!!!????!.xxxxxxxx – ! indicates matched digits, ? indicates no
match, x indicates not checked (beyond subnet mask)

11000000.10101000.00001010.00000010 – Destination address for packet
11000000.10101000.00001010.00000000 – Interface Ethernet1, 24 bit subnet mask
!!!!!!!!.!!!!!!!!.!!!!!!!!.xxxxxxxx – ! indicates matched digits, ? indicates no
match, x indicates not checked (beyond subnet mask)
```

The entry for Ethernet0 matches the first 19 bits fine, but then stops matching. That means it's not the proper destination interface. You can see that the interface Ethernet1 matches 24 bits of the destination address. Ah, ha! The packet is bound for interface Ethernet1.

In a real-life router, the routing table is sorted in such a manner that the longest subnet masks are checked for matches first (i.e. the most specific routes), and numerically so that as soon as a match is found the packet can be routed and no further matching attempts are necessary (meaning that 192.168.10.0 would be listed first and 192.168.20.0 would never have been checked). Here, we're simplifying that a bit. Fancy data structures and algorithms make faster IP routers, but simple algorithms will produce the same results.

Static Routes

Up to this point, we've talked about our hypothetical router as having networks directly connected to it. That's not, obviously, how the world really works. In the pizza-driving analogy, sometimes the driver isn't allowed any further into the building than the front desk, and has to hand-off the pizza to

somebody else for delivery to the final recipient (suspend your disbelief and bear with me while I stretch my analogy, please).

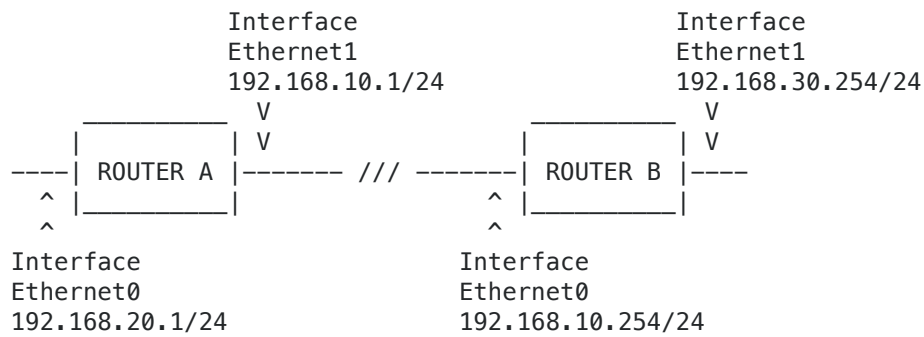
Let's start by calling our router from the earlier examples "Router A". You already know RouterA's routing table as:

- Network ID: 192.168.20.0 (11000000.10101000.00010100.00000000) - subnet mask /24 - Interface RouterA-Ethernet0
- Network ID: 192.168.10.0 (11000000.10101000.00001010.00000000) - subnet mask /24 - Interface RouterA-Ethernet1

Suppose that there's another router, "Router B", with the IP addresses 192.168.10.254/24 and 192.168.30.1/24 assigned to its Ethernet0 and Ethernet1 interfaces. It has the following routing table:

- Network ID: 192.168.10.0 (11000000.10101000.00001010.00000000) - subnet mask /24 - Interface RouterB-Ethernet0
- Network ID: 192.168.30.0 (11000000.10101000.00011110.00000000) - subnet mask /24 - Interface RouterB-Ethernet1

In pretty ASCII art, the network looks like this:



You can see that Router B knows how to "get to" a network, 192.168.30.0/24, that Router A knows nothing about.

Suppose that a PC with the IP address 192.168.20.13 attached to the network connected to router A's Ethernet0 interface sends a packet to Router A for delivery. Our hypothetical packet is destined for the IP address 192.168.30.46, which is a device attached to the network connected to the Ethernet1 interface of Router B.

With the routing table shown above, neither entry in Router A's routing table matches the destination 192.168.30.46, so Router A will return the packet to the sending PC with the message "Destination network unreachable".

To make Router A "aware" of the existence of the 192.168.30.0/24 network, we add the following entry to the routing table on Router A:

- Network ID: 192.168.30.0 (11000000.10101000.00011110.00000000) - subnet mask /24 - Accessible via 192.168.10.254

In this way, Router A has a routing table entry that matches the 192.168.30.46 destination of our example packet. This routing table entry effectively says "If you get a packet bound for 192.168.30.0/24, send it on to 192.168.10.254 because he knows how to deal with it." This is the analogous "hand-off the pizza at the front desk" action that I mentioned earlier-- passing the packet on to somebody else who knows how to get it closer to its destination.

Adding an entry to a routing table "by hand" is known as adding a "static route".

If Router B wants to deliver packets to the 192.168.20.0 subnet mask 255.255.255.0 network, it will need an entry in its routing table, too:

- Network ID: 192.168.20.0 (11000000.10101000.00010100.00000000) - subnet mask /24 - Accessible via: 192.168.10.1 (Router A's IP address in the 192.168.10.0 network)

This would create a path for delivery between the 192.168.30.0/24 network and the 192.168.20.0/24 network across the 192.168.10.0/24 network between these routers.

You always want to be sure that routers on both sides of such an "interstitial network" have a routing table entry for the "far end" network. If router B in our example didn't have a routing table entry for "far end" network 192.168.20.0/24 attached to router A our hypothetical packet from the PC at 192.168.20.13 would get to the destination device at 192.168.30.46, but any reply that 192.168.30.46 tried to send back would be returned by router B as "Destination network unreachable." One-way communication is generally not desirable. Always be sure you think about traffic flowing in both directions when you think about communication in computer networks.

You can get a lot of mileage out of static routes. Dynamic routing protocols like EIGRP, RIP, etc, are really nothing more than a way for routers to exchange routing information between each other that could, in fact, be configured with static routes. One large advantage to using dynamic routing protocols over static routes, though, is that dynamic routing protocols can dynamically change the routing table based on network conditions (bandwidth utilization, an interface "going down", etc) and, as such, using a dynamic routing protocol can result in a configuration that "routes around" failures or bottlenecks in the network infrastructure. (Dynamic routing protocols are WAY outside the scope of this answer, though.)

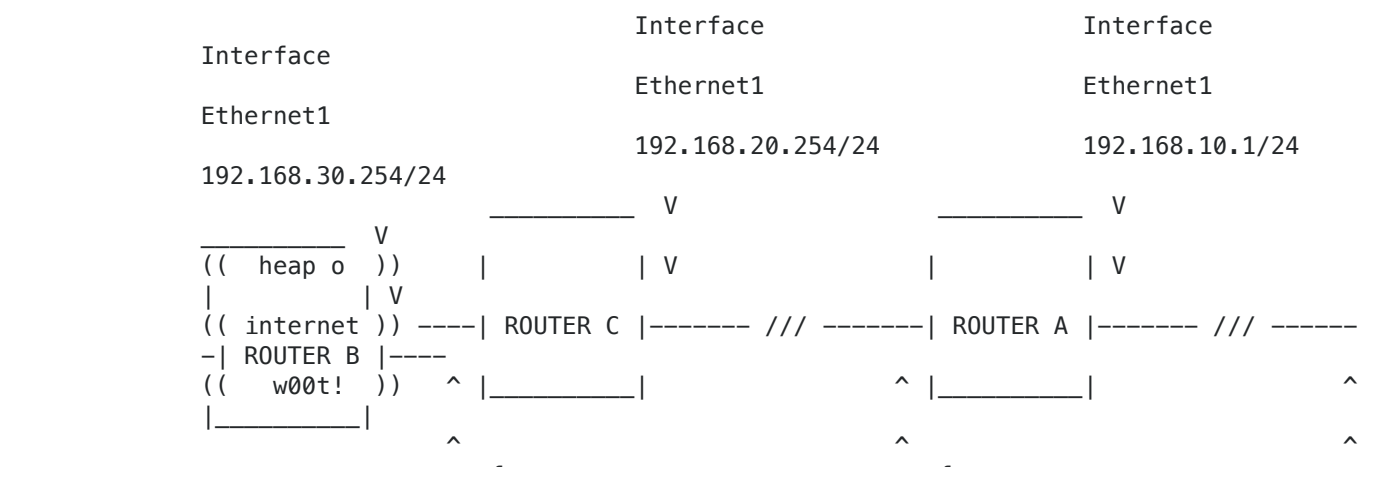
You Can't Get There From Here

In the case of our example Router A, what happens when a packet bound for "172.16.31.92" comes in?

Looking at the Router A routing table, neither destination interface or static route matches the first 24 bits of 172.18.31.92 (which is 10101100.00010000.00011111.01011100, BTW).

As we already know, Router A would return the packet to the sender via a "Destination network unreachable" message.

Say that there's another router (Router C) sitting at the address "192.168.20.254". Router C has a connection to the Internet!



```
               Interface                    Interface
Interface
               Ethernet0                    Ethernet0
Ethernet0

               10.35.1.1/30                 192.168.20.1/24
192.168.10.254/24
```

It would be nice if Router A could route packets that do not match any local interface up to Router C such that Router C can send them on to the Internet. Enter the "default gateway" route.

Add an entry at the end of our routing table like this:

- Network ID: 0.0.0.0 (00000000.00000000.00000000.00000000) - subnet mask /0 - Destination router: 192.168.20.254

When we attempt to match "172.16.31.92" to each entry in the routing table we end up hitting this new entry. It's a bit perplexing, at first. We're looking to match zero bits of the destination address with... wait... what? Matching zero bits? So, we're not looking for a match at all. This routing table entry is saying, basically, "If you get here, rather than giving up on delivery, send the packet on to the router at 192.168.20.254 and let him handle it".

192.168.20.254 is a destination we DO know how to deliver a packet to. When confronted with a packet bound for a destination for which we have no specific routing table entry this "default gateway" entry will always match (since it matches zero bits of the destination address) and gives us a "last resort" place that we can send packets for delivery. You'll sometimes hear the default gateway called the "gateway of last resort."

In order for a default gateway route to be effective it must refer to a router that is reachable using the other entries in the routing table. If you tried to specify a default gateway of 192.168.50.254 in Router A, for example, delivery to such a default gateway would fail. 192.168.50.254 isn't an address that Router A knows how to deliver packets to using any of the other routes in its routing table, so such an address would be ineffective as a default gateway. This can be stated concisely: The default gateway must be set to an address already reachable by using another route in the routing table.

Real routers typically store the default gateway as the last route in their routing table such that it matches packets after they've failed to match all other entries in the table.

Urban Planning and IP Routing

Breaking up a IP subnet into smaller IP subnets is lke urban planning. In urban planning, zoning is used to adapt to natural features of the landscape (rivers, lakes, etc), to influence traffic flows between different parts of the city, and to segregate different types of land-use (industrial, residential, etc). IP subnetting is really much the same.

There are three main reasons why you would subnet a network:

- You may want to communicate across different unlike communication media. If you have a T1 WAN connection between two buildings IP routers could be placed on the ends of these connections to facilitate communication across the T1. The networks on each end (and possibly the "interstitial" network on the T1 itself) would be assigned to unique IP subnets so that the routers can make decisions about which traffic should be sent across the T1 line.
- In an Ethernet network, you might use subnetting to limit the amount of broadcast traffic in a given portion of the network. Application-layer protocols use the broadcast capability of Ethernet for very useful purposes. As you get more and more hosts packed into the same Ethernet network, though, the percentage of broadcast traffic on the wire (or air, in wireless Ethernet) can increase to such a point as to create problems for delivery of non-broadcast

traffic. (In the olden days, broadcast traffic could overwhelm the CPU of hosts by forcing them to examine each broadcast packet. That's less likely today.) Excessive traffic on switched Ethernet can also come in form of "flooding of frames to unknown destinations". This condition

is caused by an Ethernet switch being unable to keep track of every destination on the network and is the reason why switched Ethernet networks can't scale to an infinite number of hosts. The effect of flooding of frames to unknown destinations is similar to the the effect of excess broadcast traffic, for the purposes of subnetting.

- You may want to "police" the types of traffic flowing between different groups of hosts. Perhaps you have print server devices and you only want authorized print queuing server computers to send jobs to them. By limiting the traffic allowed to flow to the print server device subnet users can't configure their PCs to talk directly to the print server devices to bypass print accounting. You might put the print server devices into a subnet all to themselves and create a rule in the router or firewall attached to that subnet to control the list of hosts permitted to send traffic to the print server devices. (Both routers and firewalls can typically make decisions about how or whether to deliver a packet based on the source and destination addresses of the packet. Firewalls are typically a sub-species of router with an obsessive personality. They can be very, very concerned about the payload of packets, whereas routers typically disregard payloads and just deliver the packets.)

In planning a city, you can plan how streets intersect with each other, and can use turn-only, one-way, and dead-end streets to influence traffic flows. You might want Main Street to be 30 blocks long, with each block having up to 99 buildings each. It's pretty easy to plan your street numbering such that each block in Main Street has a range of street numbers increasing by 100 for each block. It's very easy to know what the "starting number" in each subsequent block should be.

In planning IP subnets, you're concerned with building the right number of subnets (streets) with the right number of available host ID's (building numbers), and using routers to connect the subnets to each other (intersections). Rules about allowed source and destination addresses specified in the routers can further control the flow of traffic. Firewalls can act like obsessive traffic cops.

For the purposes of this answer, building our subnets is our only major concern. Instead of working in decimal, as you would with urban planning, you work in binary to describe the bounds of each subnet.

Continued on: How does IPv4 Subnetting Work?

(Yes ... we reached the maximum size of an answer (30000 characters).)

edited Apr 13 '17 at 12:14
Community ♦
1

answered Aug 4 '09 at 15:51

Evan Anderson
137k 15 178 316

142

Continued from: How does IPv4 Subnetting Work?

Your ISP gives you the range the network ID 192.168.40.0/24 (11000000.10101000.00101000.00000000). You know that you'd like to use a firewall / router device to limit communication between different parts of your network (servers, client computers, network equipment) and, as such,you'd like to break these various parts of your network up into IP subnets (which the firewall / router device can then route between).

You have:

- 12 server computers, but you might get up to 50% more
- 9 switches

- 97 client computers, but you might get more

What's a good way to break up 192.168.40.0/24 into these pieces?

Thinking in even powers of two, and working with the larger numbers of possible devices, you can come up with:

- 18 server computers - Next largest power of two is 32
- 9 switches - Next largest power of two is 16
- 97 client computers - Next largest power of two is 128

In a given IP subnet, there are two addresses reserved that can't be used as valid device IP addresses-- the address with all zeros in the host ID portion and the address with all ones in the host ID portion. As such, for any given IP subnet, the number of host addresses available is two to the power of the quantity of 32 minus the number of bits in the subnet mask, minus 2. So, in the case of 192.168.40.0/24 we can see that the subnet mask has 24 bits. That leaves 8 bits available for host IDs. We know that 2 to the 8th power is 256-- meaning that 256 possible combinations of bits fit into a slot 8 bits wide. Since the "11111111" and "00000000" combinations of those 8 bits aren't allowable for host IDs, that leaves us with 254 possible hosts that can be assigned in the 192.168.40.0/24 network.

Of those 254 hosts, it looks like we can fit the client computers, switches, and server computers into that space, right? Let's try.

You have 8 bits of subnet mask to "play with" (the remaining 8 bits of the IP address 192.168.40.0/24 not covered by the subnet mask provided by your ISP). We have to work out a way to use those 8 bits to create a number of unique network IDs that can accommodate the devices above.

Start with the largest network - the client computers. You know that the next larger power of two from the number of possible devices is 128. The number 128, in binary, is "10000000". Fortunately for us, that fits into the 8 bit slot we have free (if it didn't, that would be an indication that our starting subnet is too small to accommodate all our devices).

Let's take our network ID, as provided by our ISP, and add a single bit of subnet mask to it, breaking it up into two networks:

```
11000000.10101000.00101000.00000000 – 192.168.40.0 network ID
11111111.11111111.11111111.00000000 – Old subnet mask (/24)

11000000.10101000.00101000.00000000 – 192.168.40.0 network ID
11111111.11111111.11111111.10000000 – New subnet mask (/25)

11000000.10101000.00101000.10000000 – 192.168.40.128 network ID
11111111.11111111.11111111.10000000 – New subnet mask (/25)
```

Look over that until it makes sense. We increased the subnet mask by one bit in length, causing the network ID to cover one bit that would have been used for host ID. Since that one bit can be either zero or one, we've effectively split our 192.168.40.0 network into two networks. The first valid IP address in the 192.168.40.0/25 network will be the first host ID with a "1" in the right-most bit:

```
11000000.10101000.00101000.00000001 – 192.168.40.1 – First valid host in the
192.168.40.0/25 network
```

The first valid host in the 192.168.40.128 network will, likewise, be the first host ID with a "1' in the right-most bit:

```
11000000.10101000.00101000.10000001 – 192.168.40.129 – First valid host in the
192.168.40.128/25 network
```

The last valid host in each network will be the host ID with every bit except the right-most bit set to "1":

```
11000000.10101000.00101000.01111110 – 192.168.40.126 – Last valid host in the
192.168.40.0/25 network
11000000.10101000.00101000.11111110 – 192.168.40.254 – Last valid host in the
192.168.40.128/25 network
```

So, in this way, we've created a network large enough to hold our client computers, and a second network that we can then apply the same principle to break down into yet smaller networks. Let's make a note:

- Client computers - 192.168.40.0/25 - Valid IPs: 192.168.40.1 - 192.168.40.126

Now, to break down the second network for our servers and switches, we do the same thing.

We have 12 server computers, but we might buy up to 6 more. Let's plan on 18, which leaves us the next highest power of 2 as 32. In binary, 32 is "100000", which is 6 bits long. We have 7 bits of subnet mask left in 192.168.40.128/25, so we have enough bits to continue "playing". Adding one more bit of subnet mask gives us two more networks:

```
11000000.10101000.00101000.10000000 – 192.168.40.128 network ID
11111111.11111111.11111111.10000000 – Old subnet mask (/25)

11000000.10101000.00101000.10000000 – 192.168.40.128 network ID
11111111.11111111.11111111.11000000 – New subnet mask (/26)
11000000.10101000.00101000.10000001 – 192.168.40.129 – First valid host in the
192.168.40.128/26 network
11000000.10101000.00101000.10111110 – 192.168.40.190 – Last valid host in the
192.168.40.128/26 network

11000000.10101000.00101000.11000000 – 192.168.40.192 network ID
11111111.11111111.11111111.11000000 – New subnet mask (/26)
11000000.10101000.00101000.11000001 – 192.168.40.193 – First valid host in the
192.168.40.192/26 network
11000000.10101000.00101000.11111110 – 192.168.40.254 – Last valid host in the
192.168.40.192/26 network
```

So, now we've broken up 192.168.40.128/25 into two more networks, each of which has 26 bits of subnet mask, or a total of 62 possible host IDs-- 2 ^ (32 - 26) - 2.

That means that both of those networks have enough addresses for our servers and switches! Let's make notes:

- Servers - 192.168.40.128/26 - Valid IPs: 192.168.40.129 - 192.168.40.190
- Switches - 192.168.40.192/26 - Valid IPs: 192.168.40.193 - 192.168.40.254

This technique is called variable-length subnet masking (VLSM) and, if properly applied, causes "core routers" to have smaller routing tables (through a process called "route summarization"). In the case of our ISP in this example, they can be totally unaware of how we've subnetted 192.168.40.0/24. If their router has a packet bound for 192.168.40.206 (one of our switches), they need only know to pass it to our router (since 192.168.40.206 matches the network id and subnet mask 192.168.40.0/24 in their router's routing table) and our router will get it to the destination. This keeps our subnet routes out of their routing tables. (I'm simplifying here, but you get the idea.)

You can plan very geographically large networks in this same way. As long as you do the right "urban planning" up front (anticipating the number of hosts in each sub-network with some accuracy and an

eye to the future) you can create a large routing hierarchy that, at the core routers, "summarizes" to a very small number of routes. As we saw above, the more routes that are in a router's routing table the slower it performs its job. Designing an IP network with VLSM and keeping routing tables small is a Good Thing(tm).

The Unrealism of Examples

The fictional world in this answer is, obviously, fictional. Typically you can make subnets on modern switched Ethernet with more hosts than 254 (traffic profile dependent). As has been pointed out in comments, using /24 networks between routers isn't consistent with Real Life(tm). It makes for cute examples, but is a waste of address space. Typically, a /30 or a /31 (see http://www.faqs.org/rfcs/rfc3021.html for details on how /31's work-- they are beyond the scope of this answer for sure) network is used on links that are strictly point-to-point between two routers.

edited Apr 13 '17 at 12:14

community wiki
5 revs, 4 users 96%
Joseph Kern

Sub-netting

76

Sub-netting is not difficult but it can be intimidating. So let's start with the simplest possible step. Learning to count in binary.

Binary

Binary is a base 2 counting system. Consisting of only two numbers (1 and 0). Counting proceeds in this manner.

```
1 = 001 ( 0 + 0 + 1 = 1)
2 = 010 ( 0 + 2 + 0 = 2)
3 = 011 ( 0 + 2 + 1 = 3)
4 = 100 ( 4 + 0 + 0 = 4)
5 = 101 ( 4 + 0 + 1 = 5)
```

So if you just imagine that each 1 is a place holder for a value (all binary values are powers of two)

```
1     1     1     1     1 = 31
16  + 8  +  4  +  2  +  1 = 31
```

So... 100000 = 32. And 10000000 = 128. AND 11111111 = 255.

When I say, "I have a subnet mask of 255.255.255.0", I really mean, "I have a subnet mask of 11111111.11111111.11111111.00000000." We use subnets as a short hand.

The periods in the address, separate every 8 binary digits (an octet). This is why IPv4 is known as a 32bit (8*4) address space.

Why Subnet?

IPv4 addresses (192.168.1.1) are in short supply. Sub-netting gives us a way to increase the amount of available networks (or hosts). This is for administrative reasons and technical reasons.

Each IP address is broken into two separate portions, the network and the host. By default a Class C address (192.168.1.1) uses the first 3 octets (192.168.1) for the network portion of the address. and

the 4th octet (.1) as the host portion.

By default an ip address and subnet mask for a Class C adress looks like this

```
IP     192.168.1.1
Subnet 255.255.255.0
```

In binary like this

```
IP     11000000.10101000.00000001.00000001
Subnet 11111111.11111111.11111111.00000000
```

Look at the binary example again. Notice how I said the first three octets are used for the network? Notice how the network portion is all ones? That's all sub-netting is. Let's expand.

Given that I have a single octet for my host portion (in the above example). I can ONLY ever have 256 hosts (256 is the max value of an octet, counting from 0). But there's another small trick: you need to subtract 2 host addresses from the available ones (currently 256). The first address in the range will be for the network (192.168.1.0) and the last address in the range will be the broadcast (192.168.1.255). So you really have 254 available addresses for hosts in one network.

A Case Study

Let's say I gave you the the following piece of paper.

```
Create 4 networks with 192.168.1.0/24.
```

Let's take a look at this. The /24 is called CIDR notation. Rather than referencing the 255.255.255.0 we just reference the bits we need for the network. In this case we need 24bits (3*8) from a 32bit address. Writing this out in binary

```
11111111.11111111.11111111.00000000 = 255.255.255.0
8bits   + 8bits  + 8bits  + 0bits   = 24bits
```

Next we know we need figure out how many subnets we need. Looks like 4. Since we need to create more networks (currently we only have one) lets flip some bits

```
11111111.11111111.11111111.00000000 = 255.255.255.0   = 1 Network OR /24
11111111.11111111.11111111.10000000 = 255.255.255.128 = 2 Networks OR /25
11111111.11111111.11111111.11000000 = 255.255.255.192 = 4 Networks (remember
powers of 2!) OR /26
```

Now that we've decided on a /26 let's start allocating hosts. A little simple math:

```
32(bits) - 26(bits) = 6(bits) for host addresses.
```

We have 6bits to allocate in each network for hosts. Remembering that we need to subtract 2 for each network.

```
h = host bits
```

```
2^h - 2 = hosts available

2^6 - 2 = 62 hosts


Finally we have 62 hosts in 4 networks, 192.168.1.0/26
```

Now we need to figure out where the hosts go. Back to the binary!

```
11111111.11111111.11111111.00,000000 [the comma is the new network/hosts
division]

Begin to calculate:

11000000.10101000.00000001.00,000000 = 192.168.1.0 [First IP = Network Adress]
11000000.10101000.00000001.00,000001 = 192.168.1.1 [First Host IP]
11000000.10101000.00000001.00,000010 = 192.168.1.2 [Second Host IP]
11000000.10101000.00000001.00,000011 = 192.168.1.3 [Third Host IP]

And so on ... until ...

11000000.10101000.00000001.00,111110 = 192.168.1.62 [Sixty Second Host IP]
11000000.10101000.00000001.00,111111 = 192.168.1.63 [Last IP = Broadcast
Address]

So ... On to the NEXT network ....

11000000.10101000.00000001.01,000000 = 192.168.1.64 [First IP = Network Address]
11000000.10101000.00000001.01,000001 = 192.168.1.65 [First Host IP]
11000000.10101000.00000001.01,000010 = 192.168.1.66 [Second Host IP]

And so on ... until ...

11000000.10101000.00000001.01,111110 = 192.168.1.126 [Sixty Second Host IP]
11000000.10101000.00000001.01,111111 = 192.168.1.127 [Last IP = Broadcast
Address]

So ... On to the NEXT network ....

11000000.10101000.00000001.10,000000 = 192.168.1.128 [First IP = Network
Address]
11000000.10101000.00000001.10,000001 = 192.168.1.129 [First Host IP]

Etc ...
```

In this way you can calculate the entire subnet.

Wild Cards A wild card mask is an inverted subnet mask.

```
11111111.11111111.11111111.11000000 = 255.255.255.192 [Subnet]
00000000.00000000.00000000.00111111 = 0.0.0.63 [Wild Card]
```

Further

Google for the terms 'super-netting', and 'VLSM (variable length subnet mask)', for more advanced topics.

I can see now that I took too long in responding ... **sigh**

edited Nov 30 '11 at 15:53

answered Aug 4 '09 at 15:58

Joseph Kern
9,298 2 27 55

34

A brief history lesson: originally, unicast IPv4 addresses were divided into 3 classes, each with an associated 'default' mask length (called the classful subnet mask)

- Class A: Anything in the range 1.0.0.0 -> 127.255.255.255. Classful subnet mask of 255.0.0.0 (/8 in CIDR notation)
- Class B: Anything in the range 128.0.0.0 -> 191.255.255.255. Classful subnet mask of 255.255.0.0 (/16 in CIDR notation)
- Class C: Anything in the range 192.0.0.0 -> 223.255.255.255. Classful subnet mask of 255.255.255.0 (/24 in CIDR notation)

The idea was that different sized organisations could be allocated a different class of IP address, to make efficient use of IP address space.

However, as IP networks grew, it became clear that this approach had its problems. To name but three:

In a classful world, all subnets **had** to have a mask of /8, /16, or /24. This meant that the smallest subnet that could be configured was a /24, which allowed for 254 host addresses (.0 and .255 being reserved as the the network and broadcast addresses, respectively). This was tremendously wasteful, particularly on point-to-point links with only two routers attached to them.

Even after this restriction was relaxed, earlier routing protocols (e.g. RIPv1) did not advertise the mask length associated with an IP prefix. In the absence of a specific mask, it would use either the mask of a directly connected interface in the same classful network, or fall back to using the classful mask. For example, if you wanted to use the network 172.16.0.0 for inter-router links with /30 masks, **all** subnets from 172.16.0.0 - 172.16.255.255 would have to have a /30 mask (16384 subnets, each with 2 useable IPs).

The routing tables of internet routers began to take up more and more memory; this was/is known as the 'routing table explosion'. If a provider had 16 contiguous /24 networks, for example, they would need to advertise all 16 prefixes, rather than a single summary that covered the entire range.

Two related refinements allowed us to move beyond the above limitations.

1. Variable Length Subnet Masks (VLSM)
2. CIDR (Classless inter domain routing)

VLSM refers to the ability of a routing protocol to support different subnet masks within the same classful network. For example:

```
192.168.1.0/24
```

Could be split into:

```
192.168.1.0/25
192.168.1.128/26
192.168.1.192/27
192.168.1.224/27
```

Which allowed for much more efficient use of address space; subnets could be sized correctly for the number of hosts/routers that would be attached to them.

CIDR takes VLSM and extends it the other way; in addition to splitting a single classful network into

smaller subnets, CIDR allows for the aggregation of multiple classful networks into a single summary. For example, the following Class B (/16) networks:

```
172.16.0.0/16
172.17.0.0/16
172.18.0.0/16
172.19.0.0/16
```

Can be aggregate/summarised with a single prefix:

```
172.16.0.0/14
```

In terms of subnetting: a subnet mask is 32 bits long. The mask length denotes how many bits identify the network portion of the address. For example:

```
10.1.1.0/24
```

- The classful subnet mask is /8
- The actual subnet mask is /24
- 16 bits (24-8) have been 'borrowed' for the use of subnetting.

This means that, assuming the entire 10.0.0.0/8 network is subnetted into /24s, that there will be 65536 (2 ^ 16) subnets within this range. (This is assuming that the platform you are using supports subnet numbers of 0 and 255. See Cisco's ip subnet-zero).

There are 8 bits remaining in the 'host portion' of the address. This means there are 256 available IP addresses (2 ^ 8), of which 2 are reserved (10.1.1.0 is the network address, 10.1.1.255 is the subnet directed broadcast address). This leaves 254 usable IP addresses on this subnet. ((2^8) - 2)

answered Aug 4 '09 at 15:53

Murali Suriar

8,581 8 36 59

7

Network ranges: networks are always referenced by 2 numbers: one to determine the network, and another to determine which computer (or host) is on that network. As each nertwork address is 32 bits long, both numbers have to fit in these 32 bits.

Network numbering is important, as its this that ICANN hands out when you ask for a network IP range. If we didn't have it, no-one would be able to tell the difference between my network and AT&Ts. So while these numbers must be unique, no-one else wants to assign numbers to the hosts that are on my network. Hence the split - the first part is managed by the network people, the second part is all mine to give to whatever machines I want.

The network number isn't fixed at a certain number of bits - for example, if I had only 200 machines to manage myself, I'd be perfectly happy with a network number that used 24 bits, leaving me with only 8 bits for myself - which is enough for up to 255 hosts. As the network number uses 24 bits, we can have lots of them, meaning lots of people can have their own networks.

In the past this was referred to as a class C network. (class B used 16 bits for network number, and class A used 8 bits, so there are only a few class A networks in existence).

Nowadays, this naming convention has fallen out of fashion. It was replaced with the concept called CIDR. CIDR explicitly puts the number of bits for your hosts after the slash. So my example above

CIDR. CIDR explicitly puts the number of bits for your hosts after the slash. So my example above (the class C) is now referred to as a CIDR /24.

This does give us a little more flexibility, before if I had 300 hosts to manage, I'd need a class B network! Now, I can just get a /23 CIDR, so I have 9 bits for me, and 23 bits for the network number. ICANN may not have these kind of networks out, but if I have an internal one, or am renting a partial network from an ISP, this makes it easier to manage - especially as all their customers can be given a /29 (leaving me .. 3 bits or a maximum of 8 machines) which allows more people to have their own little slice of the available IP addresses. Until we get IPv6, this is quite important.

However... while I know a /24 CIDR is the equivalent of the old Class C network, and a /16 is class B and a /8 is a class A... I am still stumped trying to calculate a /22 in my head. Fortunately there are tools that do this for me :)

However - if you know a /24 is 8 bits for hosts (and 24 bits for network), then I know a /23 gives me an extra bit which doubles the number of hosts.

answered Aug 4 '09 at 15:12

gbjbaanb
3,662 1 19 27

5

I'll pose and answer a few related questions along the way:

- Why do you see `255.255.255.0` so often?
- Why `192.168.0.1` ?
- Why `127.0.0.1` ?

Why such weird numbers — 255, 192, 168, 127?

8+8+8+8-bit dotted decimal

Internet addresses like 194.60.38.10 use **dotted-decimal notation** to split 32 bits up into 8+8+8+8 bits. Dotted-decimal means converting† each number to binary then left-padding it with `0` 's.

For example `.60.` → 60=32+16+8+4 → `111100` → `.00111100.` .

So 194.60.38.10 is dotted-decimal for the 4×8=32-bit address `11000010.00111100.00100110.00001010` , since 38 → `100110` , 10 → `1010` , and so on. 194 requires all 8 bits; the rest are padded.

Once you think about 255, 192, and 127 in 8-bit binary, you can more easily understand why certain decimal numbers are so common:

- 255 = `11111111`
- 192 = `11000000`
- 127 = `_1111111`
- 128 = `10000000`

These decimal numbers happen to represent visually convenient 8-bit blocks like ■■■■■■■■, ■□□□□□□□, and □■■■■■■■. So you've never seen 256=2^9 because of the 8-bit limit, and 127=128−1=2^8−1 is the bit-flip of a power-of-two—and powers-of-two are `10……00000` 's in binary.

- 168 = `10101000`

Subnet masks: What's mine is mine + What's yours is yours

Subnet masks then break each 32-bit internet address up into a network ID and a host ID. Whereas internet addresses can have any mixture of 1's and 0's, subnet masks begin with only 1's and end with only 0's.

```
■■□□□□■□|□□■■■■□□|□□■□□■■□|□□□□■□■□ IP
■■■■■■■■|■■■■■■■■|■■■■■■■■|□□□□□□□□ subnet
```

Blacking out the first 8+8+8=24 bits and whiting out the final 8 bits is a way of splitting the IP ■■□□□□■□|□□■■■■□□|□□■□□■■□|□□□□■□■□ into two pieces:

```
■■□□□□■□|□□■■■■□□|□□■□□■■□          network
                              □□□□■□■□ host
```

If the subnetwork owner (say OmniCorp) wanted more internal IP's, they could buy up more (say 8+8=16 bits) of the righthand side of the network, like this:

```
■■□□□□■□|□□■■■■□□|□□■□□■■□|□□□□■□■□ IP
■■■■■■■■|■■■■■■■■|□□□□□□□□|□□□□□□□□ subnet
■■□□□□■□ □□■■■■□□                    network
                  □□■□□■■□ □□□□■□■□ host
```

Clearly there is a tradeoff within the 32-bit = 2^{32} = 4,294,967,296-option address space: if you buy up more network ID's (lefthand side) your internal network has more host ID's (righthand side) to assign.

Cheap people therefore have a subnet mask of

255.255.255.0 = ■■■■■■■■|■■■■■■■■|■■■■■■■■|□□□□□□□□.

Even cheaper people have

255.255.255.128 = ■■■■■■■■|■■■■■■■■|■■■■■■■■|■□□□□□□□

or 255.255.255.192 = ■■■■■■■■|■■■■■■■■|■■■■■■■■|■■□□□□□□.

According to folklore, it wasn't actually Roger Miller, but a lowly sysadmin with a 255.255.255.254 mask who originally wrote King of the Road, substituting "I ain't got a large subnet" for "I ain't got no cigarettes".

(Why are the masks of the lowly filled with such high numbers? Because, like Miller's narrator, subnet masks count all the things you don't have.)

What does the trailing slash after an IP mean? (eg, 194.60.38.10/24)

Since subnet masks (which divide "theirs" from "ours") always begin with `1`'s, and since we hate summing up powers-of-two even more than we hate figuring the powers-of-two in the first place, someone invented CIDR (the slash after an IP).

194.60.38.10/24 means "the submask has 24 ones, then the rest are zeroes", so

■■■■■■■■|■■■■■■■■|■■■■■■■■|□□□□□□□□ with 8+8+8 bits belonging to "them" and 8 bits belonging to "us".

Reversing the hobo's anthem above,

- `/31` is the songwriter
- `/24` is middle-class (`255.255.255.0` = ■■■■■■■■|■■■■■■■■|■■■■■■■■|□□□□□□□□
- `/16` is rich ■■■■■■■■|■■■■■■■■|□□□□□□□□|□□□□□□□□
- `/8` is super rich ■■■■■■■■|□□□□□□□□|□□□□□□□□|□□□□□□□□
- `/1` or `/0` would be the IANA or something.

† Use `bc -l; obase=10; 60` for example.

edited Aug 4 at 15:09

community wiki
2 revs, 2 users 94%
isomorphismes

4

While the above is correct (sorry, TL;DR), calculating subnets still causes many network administrators a lot of grief. There actually is a very easy way to do subnet calculation, you can do most of it in your head, and there is very little you have to memorize. For most applications, it's not even necessary to understand the binary representation, though it is helpful for a complete understanding of subnetting. Here I will only discuss IPv4; IPv6 is outside of the scope of this discussion.

Remember this:

There are three key things to remember: all subnets are based on powers of two, and there are two key numbers: 256 and 32. More on that later.

First, lets look at a table containing powers of 2:

```
2^0 = 1
2^1 = 2
2^2 = 4
2^3 = 8
2^4 = 16
2^5 = 32
2^6 = 64
2^7 = 128
2^8 = 256
```

Calculating powers of 2 is easy: each integer increase in the power doubles the result. 1+1=2, 2+2=4, 4+4=8, 8+8=16, and so on. The total number of addresses in a subnet must always be a power of 2.

Since each octet of an IPv4 subnet goes up to 256, 256 is a very important number and forms the basis for the rest of the math.

Sizing the subnet

We'll start with an easy question: "how many addresses in a subnet if the mask is 255.255.255.248?" We will ignore the first three octets for now and look at the last. Here is how easy it is: subtract 248 from 256. 256 minus 248 equals 8. There are 8 addresses available (including the network and broadcast addresses). The reverse also works: "if I want to have a subnet with 16 addresses, what will the subnet mask be?" 256 minus 16 equals 240. The subnet mask will be 255.255.255.248.

Now if we want to expand beyond 256 addresses (historically, a "class C"), it gets only a tiny bit more complicated: if our last octet is 0 and our third octet is, say, 240, (255.255.240.0) then we do the math on the third octet and find that there would be 16 addresses. So we multiply 16 by 256 (the number of addresses in the last octet) to get 4,096. If both the last two octets are 0, (ex. 255.240.0.0) then we take the subtraction result from the second octet (we'll say it's 16 again), multiply but 256 (addresses in the third octet), multiply again by 256 (addresses in the last octet) to get 1,048,576 addresses. Easy as that! (OK, so the reverse is a little more difficult. If we want a subnet with

1,048,576 addresses, we'll have to divide that number by 256 a couple of times to get a number we can subtract from 256.)

Network address

Now that we know how to calculate the subnet mask, how do we figure out what the network address is? That's easy: it's always a multiple of the number of addresses in our subnet. So if we have 16 addresses in our subnet, the possible network addresses will be 0, 16, 32, 48, 64, and so on up to 240. (Note that 0 is a valid multiple of any number, as any number multiplied by 0 equals 0.)

And, of course, the broadcast address will be the last address in the scope. So if we have 16 address in our subnet, and we've chosen a network address of 10.3.54.64, the broadcast address will be (64+16-1=79) 10.3.54.79.

CIDR Notation

So how about CIDR notation? How do translate that to and from an IPv4-style subnet mask?

Remember our powers of two? Well, now we have another key number to remember besides 256: 32. Remember, CIDR notation describes the number of significant bits in the IPv4 address, and there are 32 bits in an IPv4 address, 8 for each octet. So if we have a subnet mask of 255.255.255.240, that is 16 addresses. If we look at our "powers of 2" table above, we see that 16 is two to the fourth power (2^4). So we subtract that power number -- 4 -- from 32 and get 28. Our CIDR notation for a subnet mask of 255.255.255.240, our CIDR notation is /28.

And if we are given a CIDR of /28, we subtract that (28) from 32 to get 4; raise 2 to that (4th) power (2^4) to get 16; then subtract that (16) from 256 to get 240; or 255.255.255.240.

answered Mar 14 '15 at 5:51 community wiki
Jonathan J

2

I also feel that there should atleast be a mention of NATs, because they are used so commonly in modern networks in place of Subnets, because of IPv4 address exhaustion,among other things. (Also, when I was first learning about subnets, I was very confused as to how subnetting relates to the networks created by WiFi routers).

NAT (network address translation) is a technique (commonly) used to create private networks by mapping one address space (IP:Port) to another. Majorly, this is used to create a private network of multiple private IPs behind one public address, for example, in Wifi routers, by organizations(like a university or a corporation), or sometimes by ISPs.

The actual address translation is done **transparently** in NAT capable nodes, usually routers. It may be of many forms, Full Cone, Address Restricted, Port restricted etc. or a mixture of these, which dictates how the connections across the node may be initiated.

Full details can be found on Wikipedia, but for example consider a Wifi router with 2 devices connected to it. The public IP of the router is `10.9.20.21/24`, and the IP of the devices(Private IPs) are `A: 192.168.0.2`, `B: 192.168.0.3` and that of the router is `R: 192.168.0.1`. Thus if `A` wants to connect to server `S: 10.9.24.5/24`, (which is actually on a different subnet w.r.t the router here):

1. A sends an IP packet to `R` (which would be the default gateway) with the source IP `192.168.0.2`, src port (say) `14567`, and destination IP: `10.9.24.5` (Although port is actually a part of the TCP header).

2. The router(which is NAT capable) maps the port `14567` to device `A` and changes the source on the IP packet to `10.9.20.21` (which is the public IP of the router). This is in contrast to subnetting described above, **where the IP packets are actually never changed**.
3. `S` recieves the series of TCP packets(with src IP: `10.9.20.21` , src Port: `14567`) and send response packets with those values in the destination fields.
4. `R` checks the destination port, which is `14567` and forwards the packet to `A` .
5. `A` receives the response packet.

In the above situation, if `B` tried to open a connection on the same source port (`14567`), it would be mapped to a different port by `R` (and the port in the outgoing packet changed) before sending to `S` . That is, there would also be port translation instead of just IP.

Two things to note here:

1. Due to this address translation, it is often not possible to initiate a connection to devices in the private network without using some special techniques.
2. The restriction on the total TCP connections from same device to a server (65536 = 2^16) now applies collectively to all devices behind the NAT, in the NAT form used above.

edited May 2 '17 at 19:51

community wiki
4 revs
forumulator